4º Ingeniería Informática

II26 Procesadores de lenguaje

Práctica 2: Policalc

1. Objetivos

En la segunda práctica del curso vamos a seguir profundizando en el estudio de las técnicas empleadas en la primera práctica, aplicándolas a un caso en el que la complejidad del lenguaje de entrada es mayor. Para ello, vamos a implementar una calculadora que permita al usuario realizar cálculos con enteros, reales y polinomios, usar variables y controlar (de forma reducida) el flujo de ejecución mediante instrucciones de selección en las que intervienen operadores relacionales. Esto nos sitúa en un nivel de complejidad intermedio entre el intérprete MICROCALC desarrollado en la primera práctica y el compilador que desarrollaremos en la última práctica del curso.

Nos limitaremos a operar con polinomios de una variable con coeficientes reales. En el resto de este documento cuando se hable de polinomios se sobreentiende eso.

2. Invocación y comportamiento de la calculadora

La calculadora debe poder ejecutarse interactivamente escribiendo en un terminal la orden:

`./policalc [`*opción*`]`

Su labor es analizar y ejecutar cada línea que le llegue por la entrada estándar y, si fuera necesario, mostrar el resultado en la salida estándar. En caso de que la línea leída contenga un error, se mostrará en la salida de error estándar un mensaje en ASCII (sin acentos) que indique el número de línea (contando desde uno) y el tipo de error ("`Linea` *n*`: Error lexico.`", "`Linea` *n*`: Error sintactico.`", "`Linea` *n*`: Error semantico.`" o "`Linea` *n*`: Error de ejecucion.`") y se analizará la línea siguiente. Los posibles efectos secundarios de la línea no sucederían (esto es, ni se definiría variable alguna, ni se cambiaría su valor ni se escribiría nada por la salida estándar).

Con la opción `-l` debe mostrar la secuencia de componentes léxicos, con la opción `-a` debe mostrar el árbol de derivación, y con la opción `-s` debe mostrar el AST, todo ello por la salida estándar, y en el formato de la herramienta VERARBOL en el caso de los árboles. Dichas opciones son excluyentes e inhiben la salida de resultados. El comportamiento de estas opciones ante líneas de entrada erróneas está indefinido.

3. Estructura de la entrada

La entrada de POLICALC es una secuencia de líneas de tres tipos: definiciones de variables (sec. 6), sentencias (sec. 8) y líneas vacías (sec. 9). Todas las líneas del programa, sin excepción, deben terminar con el carácter salto de línea (cosa que supondremos a partir de ahora sin repetirlo en cada apartado).

Un ejemplo de programa en lenguaje POLICALC con los tres tipos de líneas es el siguiente:

```
var p, q: polinomio                              // Definición de variables
p = [11.1 * 2.2 , 11.1 * 3.3, 0, 12/5-1]         // Sentencia
q = p * 2^3 + [7.5, 0, 0, 4.6/2] * -2 * |p(5)|   // Sentencia
escribe -q                                       // Sentencia
                                                 // Línea vacía
var r: real                                      // Definición de variables
r = 5                                            // Sentencia
si q(3/2)^2 > 5/3 entonces escribe r * 4.5       // Sentencia
```

4. Algunos elementos léxicos

4.1. Identificadores y palabras reservadas

Los identificadores en POLICALC están formados por letras y dígitos, deben empezar por letra y no deben coincidir con ninguna palabra reservada. Debe tenerse en cuenta que la caja es relevante en POLICALC, es decir, que `distancia` y `Distancia` se consideran identificadores distintos.

Las siguientes palabras clave son reservadas y, por tanto, el usuario de POLICALC no puede utilizarlas como identificadores: `entero`, `entonces`, `escribe`, `polinomio`, `real`, `si`, `var`, `x`. Nótese que, al ser la caja relevante, no son palabras reservadas `Entonces` o `POLINOMIO`, por ejemplo.

4.2. Literales

En POLICALC hay tres tipos de literales:

Literales enteros: formados por secuencias de dígitos. Por ejemplo: `593` y `007`. Representan valores enteros.

Literales reales: formados por una secuencia de uno o más dígitos seguidos de un punto y uno o más dígitos. Por ejemplo, `48.55`, `23.0` o `007.700`. Representan valores reales.

Literal de polinomio: la palabra reservada `x`. Representa el polinomio x.

4.3. Componentes léxicos que se omiten

Los comentarios se abren con una secuencia de dos barras `//` y llegan hasta el final de la línea, sin incluir el carácter de salto de línea.

Los componentes léxicos pueden separarse por cualquier secuencia de espacios en blanco y tabuladores, que son omitidos.

5. Tipos de datos

El lenguaje dispone de cuatro tipos de datos: lógico, entero, real, polinomio. Se considera que el tipo entero es menos general que el real, que a su vez lo es menos que el polinomio. En los casos en los que se permite más adelante, es posible la conversión de un valor menos general a otro más general. Cuando se convierte de entero a real, se siguen las reglas habituales; la conversión de real a polinomio se hace considerando el real como un polinomio de grado cero cuyo término independiente es el real; finalmente, la conversión de un entero a polinomio sucede en dos pasos: de entero a real y de real a polinomio.

El tipo lógico no es comparable a los restantes ni se definen para él reglas de conversión.

6. Definiciones de variables

Cada línea de definición de variables tiene la siguiente sintaxis:

`var` *`identificadores`* `:` *`tipo`*

La lista *`identificadores`* consta de uno o más identificadores separados por comas diferentes entre sí y previamente sin definir, ni con este ni con ningún otro tipo. Para identificar los tipos en las definiciones, se usan las palabras reservadas `entero`, `real` y `polinomio`. No es posible definir variables de tipo lógico. Las variables de tipo entero y real se inicializan por defecto con el valor cero, y las de tipo polinomio con el polinomio nulo. No se pueden cambiar esos valores iniciales en las definiciones.

7. Expresiones

En POLICALC son expresiones válidas los identificadores de variables previamente definidas en el programa y los literales (enteros, reales o de polinomio). Además, también son válidas aquellas expresiones que pueden construirse a partir de otras utilizando correctamente los operadores ofrecidos por el lenguaje. Finalmente, cualquier expresión válida puede encerrarse entre paréntesis para formar así una nueva expresión válida.

El lenguaje dispone de los siguientes operadores, ninguno de los cuales acepta operandos de tipo lógico, aunque algunos pueden producirlos como resultado:

Operadores relacionales: `<`, `>`, `<=`, `>=`, `==` y `!=`. Son todos binarios, infijos y asociativos por la izquierda. Admiten como operandos dos expresiones de tipo real o entero. El resultado es de tipo lógico[1]. Los operadores `==` y `!=` admiten también como operandos dos expresiones de tipo polinomio. Se considera que dos polinomios son iguales si y sólo si tienen los mismos coeficientes no nulos en las mismas posiciones. En todos los casos, cuando los operandos son de tipos distintos, el de tipo menos general se convierte al del tipo más general.

Operadores aditivos: `+`, `-`. Son binarios, infijos y asociativos por la izquierda. El resultado es del tipo más general de los tipos de los operandos, al que es convertido automáticamente el operando de tipo menos general, si lo hubiere.

Operador producto: `*`. Es binario, infijo y asociativo por la izquierda. Si ambos operandos son del mismo tipo, el resultado es de ese tipo. Si uno de los operandos es de tipo real y el otro entero, el resultado es real. Si uno de los operandos es de tipo polinomio y el otro entero o real, el resultado es el polinomio que se obtiene multiplicando cada uno de los coeficientes del polinomio por el otro operando.

Operador división: `/`. Es binario, infijo y asociativo por la izquierda. El operando derecho debe ser entero o real. Si ambos operandos son del mismo tipo, el resultado es de ese tipo. Si uno de los operandos es entero y el otro real, el resultado es real. Si el operando izquierdo es de tipo polinomio, el resultado es el polinomio que se obtiene dividiendo cada uno de los coeficientes del polinomio por el otro operando. Si el operando derecho es igual a cero, se produce un error de ejecución.

Operador potencia: `^`. Es binario, infijo y asociativo por la derecha. El operando izquierdo debe ser de tipo entero, real o polinomio, mientras que el derecho debe ser de tipo entero. El resultado es el derivado de multiplicar el operando izquierdo por sí mismo el número de veces que indique el derecho, si fuera mayor que cero. Si el operando derecho es cero, el resultado es la unidad del tipo correspondiente. Si el operando derecho es menor que cero, se produce un error de ejecución.

Operador cambio de signo: `-`. Es unario y prefijo. El resultado es del mismo tipo que el operando. Si el operando es de tipo polinomio, el resultado es de tipo polinomio y se obtiene cambiando el signo a cada uno de los coeficientes.

Operador identidad: `+`. Es unario y prefijo. El resultado es el valor del operando (por tanto, tiene su mismo tipo).

Operador valor absoluto: `||`. Es unario. El operando debe ser una expresión de tipo entero o real encerrada entre las dos barras. El resultado es su valor absoluto, del mismo tipo.

Operador de construcción de polinomios: `[]`. Permite construir un polinomio $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ a partir de la lista de sus coeficientes entre corchetes y separados por

[1] Obsérvese que al ser el resultado de tipo lógico, no es posible encadenar comparaciones ya que se incurre en un error de tipos.

comas, ordenados de menor a mayor grado e incluyendo los coeficientes nulos de la siguiente forma:

`[`a_0`, `a_1`, `a_2`, ..., `a_n`]`

Por ejemplo, la entrada `[25.5, 4.6/2, 0, -7.7]` representaría el polinomio $25.5 + 2.3x - 7.7x^3$, mientras que `[0]` representaría el polinomio nulo. Cada coeficiente debe ser una expresión de tipo entero o real, y el resultado del operador es de tipo polinomio. Debe haber al menos un coeficiente. Los coeficientes enteros se transforman en reales.

Operador de evaluación: `()`. Se emplea para evaluar un polinomio en un punto mediante la siguiente sintaxis:

expresión`(`*expresión*`)`

Si la primera expresión es de tipo entero o real, se transforma en polinomio. La segunda expresión debe ser de tipo entero o real. El resultado es de tipo real. Por ejemplo, al evaluar el polinomio $P(x) = 100 + 3x + 5x^2$ en el punto $x = 7$ debemos obtener como resultado $P(7) = 100 + 3 \cdot 7 + 5 \cdot 7^2 = 366$. En POLICALC podemos conseguirlo de muchas maneras, por ejemplo:

```
var p : polinomio
p = 100+3*x+5*x^2
escribe p(7)
p = [100,3,5]
escribe p(7)
escribe [100,3,5](7)
escribe (100+3*x+5*x^2)(7)
```

El orden de prioridad de los operadores es el siguiente, de menor a mayor:

1. Relacionales.
2. Aditivos.
3. Producto y división.
4. Potencia.
5. Cambio de signo e identidad.
6. Evaluación.

En las expresiones se pueden emplear paréntesis para agrupar subexpresiones y alterar el orden de evaluación, de la forma habitual. Obsérvese que, en POLICALC, `-p(6)` equivale a `-(p(6))` y no a `(-p)(6)` (aunque el resultado final no cambie), del mismo modo que `-3^6` equivale a `(-3)^6` y no a `-(3^6)`. Por otro lado, los operadores valor absoluto y construcción de polinomios no necesitan un nivel de prioridad explícito puesto que se aplican sobre las expresiones que encierran.

8. Sentencias

Hay tres tipos de sentencias:

Sentencia de escritura. Para mostrar resultados por la salida estándar se debe utilizar la sentencia de escritura, que tiene la sintaxis siguiente:

`escribe` *expresión*

Al ejecutarse, se evalúa la *expresión* y se muestra el resultado seguido de un salto de línea. Los resultados de tipo lógico se muestran con la cadena `cierto` o `falso`, según corresponda. Los resultados de tipo entero o real se muestran en el mismo formato que en Python. Los polinomios se muestran en forma de suma de los correspondientes monomios con coeficientes no nulos ordenados por su grado de menor a mayor. El formato general de representación de los monomios es la secuencia formada por el coeficiente (en formato Python) seguido del signo `*`, una `x`, el signo `^` y el exponente. Si el coeficiente fuera uno, se suprimiría junto con el signo `*`; si el exponente fuera uno, se suprimiría junto con el signo `^`; finalmente, el término independiente se representa únicamente mediante el coeficiente. Si el polinomio fuera nulo, se escribiría únicamente un cero. Algunos ejemplos de escritura de polinomios son:

Polinomio	Representación
$3.5 + x^2$	`3.5+x^2`
0	`0`
$5 - 2x + 1.2x^3$	`5.0-2.0*x+1.2*x^3`

Sentencia de asignación. Se pueden emplear sentencias de asignación para evaluar expresiones y guardar el resultado en variables que podrán ser empleadas más adelante en expresiones. Para ello se emplea la siguiente sintaxis:

identificador `=` *expresión*

que consta de un identificador de variable, el signo `=` y una expresión. Al ejecutarse, se evalúa la expresión y el resultado se almacena en la variable que está a la izquierda del signo `=`. La variable tiene que estar declarada previamente y su tipo debe ser igual o más general que el de la expresión. En caso de que la expresión sea de un tipo menos general, se convierte al tipo de la variable. En POLICALC la asignación no es un operador que devuelva un resultado, y, por tanto, no puede formar parte como operando en expresiones. El resultado de evaluar la expresión no se muestra en la salida del programa.

Sentencia de selección. POLICALC dispone de sentencias de selección con la siguiente sintaxis:

`si` *expresión* `entonces` *sentencia*

Su ejecución consiste en evaluar la expresión, que debe ser de tipo lógico, y ejecutar la sentencia sólo si el resultado de la evaluación es cierto. La sentencia (que es única) puede ser de cualquiera de los tres tipos posibles (lo cual permite anidar instrucciones de selección).

9. Líneas vacías

Las líneas vacías (formadas sólo por el carácter salto de línea, o por componentes que se omiten y el salto de línea) se consideran correctas y no tienen ningún efecto.